Student Perception of Active Learning Methods in Software Engineering Education: An Experience Report with Client–Developer Role Inversion

Adriane da Costa Ferreira
Departamento de Ciências Exatas e Tecnológicas,
Universidade Federal do Amapá
Macapá, Amapá, Brasil
adriferreira1808@gmail.com

Paula M. L. de Oliveira Costa
Departamento de Ciências Exatas e Tecnológicas,
Universidade Federal do Amapá
Macapá, Amapá, Brasil
paulamaria.loc@gmail.com

Julio Cezar Costa Furtado
Departamento de Ciências Exatas e Tecnológicas,
Universidade Federal do Amapá
Macapá, Amapá, Brasil
furtado@unifap.br

Abstract

This paper presents an experience report whose aim is to evaluate student perception of the use of active learning methods in a mandatory Software Engineering course, rather than to propose a new pedagogical technique. To enable the evaluation, the development lifecycle areas were integrated into a simulated startup ecosystem in which students simultaneously played the roles of client and developer across distinct teams. The approach was applied to a class of 35 Computer Science students, of whom 32 completed the course and answered a questionnaire structured around the Goal-Question-Metric (GQM) framework, with triangulation between five-point Likert data and categorical content analysis. Most respondents rated the use of active learning methods positively, with the highest scores in Requirements Engineering, Process Models, and Modeling, and more moderate scores in Configuration Management and Testing.

Keywords

Software Engineering Education, Active Learning Methods, Student Perception, Role-Playing, Project-Based Learning

1 Introduction

Software Engineering (SE) demands practical competencies that are often underdeveloped in traditional, lecture-oriented instruction, producing a persistent gap between academic training and real-world industry demands [1, 4]. This gap is typically aggravated by self-contained academic projects that lack the pressure of real clients or scope changes. Without experiencing such negotiation dynamics, students tend to produce misaligned requirements and develop a false sense of technical mastery [7, 19].

The shortfall is particularly critical in Requirements Engineering (RE), where elicitation failures compromise the entire development cycle [14, 19]. Content-only instruction fails to prepare students for the discovery and resolution of conflicts, and the literature emphasizes the need to expose learners to real or realistic stakeholders and to active elicitation activities [7, 18].

To mitigate this gap and foster student autonomy, the literature consistently points to active learning methods, Project-Based Learning (PBL), and the simulation of industrial scenarios as well-established enablers [4, 15, 16]. Despite a growing body of supporting evidence, there remains a clear need for additional empirical reports documenting how students actually perceive these approaches in the classroom, in specific institutional settings, and over the course of a full semester [7, 15].

The present paper contributes to this need by reporting an experience in which the SE development lifecycle was integrated into a simulated startup ecosystem, where undergraduate students simultaneously acted as clients and developers across teams. This bidirectional role inversion, sustained throughout the entire semester, served as the pedagogical mechanism that enabled the systematic application of active learning methods and the collection of student perceptions about them. The study is guided by the following central question: *How do undergraduate students perceive the use of active learning methods in a Software Engineering course, in terms of learning, engagement, and technical preparedness?*

The experience and its lessons learned were structured through the Goal-Question-Metric (GQM) framework [3], applied to a class of 35 Computer Science students, of whom 32 completed the course and answered the final questionnaire. The instrument and the anonymized dataset are made available for replication, following recent recommendations regarding PBL studies in SE [15, 16].

2 Related Work

The literature on Software Engineering Education has long highlighted the limitations of traditional instruction in developing practical competencies, particularly in areas such as Requirements Engineering and Software Process Improvement (SPI). In response, approaches based on gamification, role-playing, project-based learning, and the simulation of industrial practices have been extensively investigated [4, 10, 15, 16].

Colares et al. [5], in a systematic review of 22 primary studies on the use of gamification in SPI programs, reported that playful strategies reduce resistance to change and increase engagement, although most of the analyzed proposals focus on corporate training rather than undergraduate education. In a follow-up study, Colares et al. [6] applied active learning methods to an undergraduate SPI course and observed a predominance of positive emotions (trust, joy, anticipation) and a high perception of practical applicability. The study was nevertheless restricted to the SPI area and did not address a full development cycle with team interaction under inverted roles.

In the field of Requirements Engineering, Tiwari et al. [18] reported an exploratory experience with Case-Based Learning, with reported gains in analytical capacity and in students' understanding of elicitation activities. Marutschke et al. [11] simulated an offshore project across separated teams to teach distributed RE, showing how

asynchronous communication shapes the difficulty of elicitation. At a broader scope, Daun et al. [7] conducted a systematic review covering studies on RE education published between 2010 and 2020 and identified the involvement of real or realistic stakeholders, together with the emphasis on elicitation as a central activity, as a consolidated trend of the past decade. The same review points out, as a gap, the lack of evidence-based pedagogical foundations regarding which instructional approaches are effective, particularly in contexts that articulate RE with other lifecycle areas. In previous work, the authors [8] proposed a student-centered approach based on competencies derived from CMMI-DEV for teaching RE, reporting an increase in motivation for autonomous research on requirements, but limited to the RE discipline in isolation.

On the specific topic of client–developer role inversion, Macedo et al. [10] proposed a collaborative role-playing approach for RE in remote education, in which students alternately took on the roles of client and developer across project cycles. The authors reported gains in communication, analytical reasoning, conflict resolution, and empathy. The approach was evaluated in two classes (advanced and beginning students) through Action Research and represents a direct precedent for the present study, although it remained constrained to RE and to a remote setting, without covering a full development cycle.

Within the ecosystem and startup paradigm, Ståhl et al. [16] characterized a PBL model with strong external stakeholder participation as an "eco-system approach" to SE education, documenting virtuous cycles of lasting impact for both students and industrial partners. Along convergent lines, Cico et al. [4] developed and validated, through a Delphi study, a framework for teaching SE through startup practice, identifying dimensions such as innovation bootcamps, the integration between lean startup principles and PBL, and the cultivation of interdisciplinary skills. Souza et al. [15] also investigated student perception of PBL applied to SE, with results that reinforce the motivational potential of such approaches.

Taken together, these studies consolidate active learning methods, role-playing, PBL, and the simulation of industrial practices as promising strategies for SE education. However, most reports concentrate on isolated lifecycle areas and on the design or validation of a single technique; studies that examine how students perceive their joint use across a full semester remain scarce. The present paper positions itself in this space, providing empirical evidence on student perception of the combined use of these methods in a mandatory SE course rather than proposing a new technique.

3 Methodology

This section describes the research design adopted to answer the central question of this work: *How do undergraduate students perceive the use of active learning methods in a mandatory Software Engineering course?*

3.1 Study Design

The report follows the case study guidelines of Runeson and Höst [13], well suited to documenting a contemporary phenomenon in its natural context, where the researcher has limited control over the variables and the focus lies on how and why participants perceive it. The setting documented here is the application, in a single Software Engineering I class, of a combination of active learning methods covering the full development lifecycle. The unit of analysis is student perception, collected at the end of the semester, regarding relevance, sufficiency, motivation, theory–practice integration, and the strengths and weaknesses of the experience.

The notion of active learning adopted in this work follows Prince [12], who characterizes it as any instructional method that engages students in meaningful activities and requires them to think about what they are doing, with project-based learning recognized as one of its established forms. Under this definition, the term does not designate a single technique but an umbrella that accommodates the distinct dynamics applied throughout the course. Each dynamic is therefore treated as an instance of a named category rather than as a generic synonym for practical work, and the correspondence between dynamics and categories is made explicit in Section 4.

The methodological approach is mixed, with a qualitative–interpretive emphasis. Quantitative data were collected through a five-point Likert scale questionnaire, and qualitative data were obtained from open-ended questions analyzed via categorical content analysis [2]. Triangulation between the two sources was adopted as the central analytical strategy to corroborate interpretations and mitigate biases inherent to each data modality [13].

On the ethical front, the study was conducted in accordance with principles of human-subjects research. All participants signed an Informed Consent Form, and the questionnaire was administered anonymously, without collecting identifiers that could trace responses to individual respondents. Participation was voluntary and was not tied to the academic evaluation of the course.

3.2 Research Questions and GQM Structure

The research questions were structured following the Goal-Question-Metric framework [3], chosen for its recognized adherence to empirical research in Software Engineering and for establishing a traceable chain between pedagogical objectives, evaluative questions, and concrete metrics of student perception. Four research questions (RQs) guide the evaluation, covering complementary dimensions of the teaching–learning process:

- **RQ1**: How do students perceive the relevance and sufficiency of the content covered in the course units delivered through active learning methods?
- **RQ2**: To what extent are the activities developed throughout the semester perceived as motivating and appropriate by the students?
- **RQ3**: What is the student perception of the method's adequacy in integrating theory and practice, considering the use of tools widely adopted by industry?
- **RQ4**: Which strengths and limitations of the approach do students identify, and to what extent do they perceive its transferability to other courses?

Each RQ corresponds to a specific Goal and Metric. G1 evaluates application and content (RQ1) through the frequency of positive responses on the Likert scale regarding the usefulness and sufficiency of the information. G2 measures engagement

(RQ2) through the frequency of positive responses regarding motivation and method adequacy. G3 assesses the effectiveness of tool integration (RQ3) through the rate of agreement on theory–practice alignment, complemented by feedback on the technological toolset. G4 identifies improvement opportunities (RQ4) through thematic categorization of open-ended qualitative responses, grouping reports into strengths and limitations of the approach.

3.3 Instrumentation

The evaluation was operationalized through a structured questionnaire, organized into complementary sections aligned with the GQM framework and administered anonymously at the end of the semester. The instrument was designed to correlate the pedagogical objectives with student perceptions, as summarized in Table 1. The codes shown in Table 1, namely DF for the background and context factors and Y for the perception blocks, derive from the GQM chain, in which each research question is tied to a goal, each goal to metrics, and each metric to instrument items denoted by these codes.

The first section, addressing background and context factors, collected background and initial motivation data from the participants, including prior experience with software development, self-assessed familiarity with the six main course topics (on a five-level ordinal scale), and perception of the relevance of Software Engineering. This information was used to characterize the class profile and to assess potential biases associated with the composition of the respondents.

The second section, addressing the adequacy of the pedagogical approach, measured student perception of that adequacy across five units of the course, namely Process Models, Requirements Engineering, Software Modeling and Design, Configuration Management, and Software Quality and Testing. Each unit was evaluated along three dimensions: (i) relevance of the content, (ii) sufficiency of the content for understanding the topic, and (iii) adequacy of the teaching method employed. Each dimension was measured by a statement on a five-point Likert scale ranging from 0 (Strongly disagree) to 4 (Strongly agree), totaling 15 items in this section. We note that the questionnaire was structured around the five units formally defined in the course syllabus; the units of Project Management, Interface Design, and Client Delivery, described in Table 2 of Section 4, were operationalized in class as extensions of the others and did not receive dedicated items in the instrument.

The third section gathered the attractiveness dimensions, also on a five-point Likert scale, covering aspects such as planning, motivation, theory–practice integration, complexity, creativity, and time adequacy. Additionally, the open-ended feedback questions investigated the perceived transferability of the approach to other courses, as well as the strengths and limitations identified by students, through a dichotomous question supplemented by qualitative reports.

3.4 Execution and Data Analysis

The study was conducted during the 2025.1 academic semester, with 35 undergraduate students regularly enrolled in the mandatory Software Engineering I course of the Computer Science program at the Universidade Federal do Amapá (UNIFAP), a public federal institution located in the Northern region of Brazil. All participants were in the intermediate phase of the program, with prior coursework in structured and object-oriented programming. During the semester, three students dropped the course for reasons unrelated to the research, leaving 32 students who completed the term. The questionnaire was administered in electronic format after the conclusion of all course activities and was answered by all 32 completing students, yielding a 100% response rate among the eligible population at the end of the semester (and 91.4% of the initial enrollment). A detailed description of the course and the activities about which perceptions were collected is provided in Section 4.

Table 1: Structure of the data collection instrument.

GQM	Goal	Topics Covered	Format
DF.1 Disturbance Factors	Background, motivation	Prior experience, familiarity with six SE topics, and perceived importance of SE	Multiple choice, ordinal, 5-pt Likert
Y.5 Adequacy	Evaluation of units	Five units rated on relevance, sufficiency, and method	15 items, 5-pt Likert
Y.6 Attractiveness	Engagement	Planning, motivation, theory–practice, time, complexity, creativity	7 items, 5-pt Likert
Y.7 and Y.8 Feedback	Transferability	Applicability to other courses, strengths, limitations, suggestions	Open-ended

Quantitative analysis was descriptive: relative frequencies were computed for each Likert-scale item and consolidated into percentages of positive responses (strong and moderate agreement), neutral responses, and negative responses (moderate and strong disagreement). Open-ended responses were processed through categorical content analysis following the procedure described by Bardin [2], organized into three stages: (i) pre-analysis, with floating reading of the 32 responses; (ii) material exploration, with identification of recording units, that is, sentences or excerpts conveying a single idea; and (iii) treatment of results, with grouping by semantic similarity into emerging categories and counting the frequency of mentions per category.

Coding was conducted independently by the researchers involved and reviewed by the research supervisor; cases of divergence were resolved through joint discussion to strengthen the reliability of categorization. This independent, between-coders review constitutes a first layer of triangulation, internal to the qualitative analysis.

A second layer, between data types, was then applied as the central analytical strategy, and its procedure is made explicit here. The triangulation was carried out unit by unit and question by question: the Likert frequencies for relevance, sufficiency, and method adequacy were compared against the categories that emerged from the content analysis of the open-ended responses, in a deliberate search for convergences and divergences. Convergence between the two sources was taken as corroboration of the

interpretation, as in the case of time scarcity, which was at once the lowest-scoring attractiveness item and the most cited weakness. Divergence, in turn, was flagged and discussed rather than averaged out, so that a favorable quantitative score could not mask a recurring qualitative complaint, nor the reverse. The validity threats arising from this design are discussed in detail in Section 6.

4 Intervention Context

This section describes the course and the activities about which the student perceptions analyzed in this work were collected. The aim is not to position the approach itself as a contribution, but to transparently document the setting in which the evaluation was conducted, enabling readers to properly situate and interpret the results presented. The setting is Software Engineering I, a mandatory course in the undergraduate Computer Science program, with a total workload of 90 hours distributed across the academic semester in weekly in-person meetings. The syllabus covers the fundamentals of SE, organized into course units that address Project Management and Process Models, Requirements Engineering, Configuration Management, Software Modeling and Design, Interface Design, Verification and Validation, and Client Delivery. Table 2 presents the sequence of these units in semester 2025.1, under standard SE terminology, together with the main artifacts produced in each one.

Table 2: Macro units of semester 2025.1, weeks of the content, and main artifacts produced.

Unit	Week	Details
Project Management and Process Models	1–3	Startup conception, lifecycle model selection, and BPM workflow modeling.
Requirements Engineering	4–6	Inter-team elicitation, functional and non-functional requirements, and Kanban backlog.
Software Configuration Management (SCM)	1–16	SCM document and continuous GitHub versioning across the semester.
Software Modeling and Design	7–9	Architecture and UML modeling (Use Case, Activity, Class) of each product.
Interface Design and Prototyping	10–11	Mockups and navigable prototypes in Figma or Canva, aligned with requirements.
Verification and Validation (V&V)	12–13	Test-case spreadsheet and cross-review of artifacts via the prototypes.
Client Delivery	14–16	Artifact integration, reflective presentation, and client-team validation, closing the inversion cycle.

4.1 Team Organization and Semester Dynamics

In the first class meeting, students were organized into nine teams (eight with four members and one with three), formed by student affinity, that is, by free association among the students themselves rather than by criteria defined by the instructor. This composition choice, adopted for the reported cycle, is later revisited as a limitation in Section 6. Each team then drafted a software proposal for a fictitious product to be developed throughout the semester, simulating a startup. The proposals were then redistributed across teams through a public in-class draw, so that each team simultaneously played two roles: client of the proposal it had originally conceived, now developed by another team, and developer of the proposal received from another team. This bidirectional inversion, sustained throughout the entire semester, was the core pedagogical mechanism analyzed in this work.

The parallel themes were kept operationally tractable by synchronizing all teams on the same lifecycle phase at any given moment. Although each team owned a distinct startup proposal, every team produced the same type of artifact in the same unit, so that the instructor could conduct a single lecture and a single dynamic per unit and then supervise nine parallel projects that differed in theme but coincided in stage. The draw that assigned proposals was public and held in class, which reinforced the perception of impartiality in the pairing and gave visibility to the client and developer roles each team would assume. This arrangement, in which the theme varies across teams while the phase and the expected deliverable remain common, is what allowed the ecosystem to be managed by a single instructor without multiplying the assessment effort by the number of themes.

As supporting tools for development, students used GitHub for code versioning and artifact storage, and GitHub Projects as a Kanban board for task management, mirroring tools widely adopted by the software industry. The formal deliverables across the semester included the process modeling (BPM), the requirements documentation and list populated in the project's Kanban, the Software Configuration Management (SCM) documentation, the UML modeling (Use Case, Activity, and Class diagrams), the test case spreadsheets, the interface prototypes, and the final reflective presentation. Artifacts were evaluated both by the instructor and by the corresponding client team, reinforcing the role-inversion dynamic.

Each course unit was structured according to a recurring pedagogical pattern: an initial *playful activity*, aimed at sensitizing students to the need for the content to be studied, followed by an *applied practice* in the startup context, in which the acquired knowledge was materialized into a concrete project artifact. This sequence was consistently adopted across all units, articulating with the ecosystem simulation. Following the definition adopted in Section 3, each dynamic corresponds to a recognized category of active learning: simulation and role play, materialized in the client and developer alternation and in the University Restaurant activity; project-based learning, embodied in the semester-long product development; playful and tangible activities, such as the Paper Airplane Factory; and peer assessment, expressed in the cross-team review of artifacts. The activities are described in the order of Table 2.

Project Management and Process Models. The initial playful activity was the "Paper Airplane Factory," in which teams first produced paper airplanes without a defined process and then with structured planning, empirically evidencing the impact of

process on productivity and making the lifecycle models (Waterfall, Incremental, and Reuse-Oriented) tangible. In the subsequent applied practice in the startup context, each team modeled its own development process using BPMN, translating the experience of the dynamic into a systematic representation of the workflow that would guide the rest of the semester.

Requirements Engineering. This unit included a simulation in which the instructor presented and progressively altered the requirements of a fictitious system for the University Restaurant, exposing students to scope volatility and to the need for negotiation with the client. Teams then conducted open elicitation interviews among themselves, with validation and refinement of the requirements documents led by the instructor. The consolidated items populated the project's initial Kanban backlog.

Software Configuration Management (SCM). Conducted transversally across the semester, this unit articulated the technical toolset with the project workflow: each team set up its GitHub repository, produced the SCM document, and proceeded to version artifacts (documents, models, spreadsheets, and prototypes) as they were generated, replicating the continuous configuration cycle adopted by the industry.

Software Modeling and Design. This unit encompassed the definition of the solution's architecture and the elaboration of UML Use Case, Activity, and Class diagrams applied to each startup's product, with artifacts refined based on feedback from the corresponding client team.

Interface Design and Prototyping. Aligned with the specified requirements and the produced modeling, each team built mockups and navigable prototypes of the product screens in Figma or Canva. The prototypes served as a communication instrument with the client team and as a basis for the test cases derived later.

Verification and Validation (V&V). V&V was conducted continuously throughout the project rather than as a one-off in-class activity. Teams specified test cases in a spreadsheet linked to the requirements and interface prototypes, cross-reviewed the artifacts of their partner teams, and adjusted the backlog according to the defects and inconsistencies identified.

Client Delivery. The cycle closed in two complementary sessions. In the first, each developer team delivered the product to its respective client team, which formally evaluated the artifacts and their adherence to the originally conceived proposal, completing the bidirectional inversion cycle. In the second, in a final reflective presentation, the teams shared the results produced and discussed lessons learned, challenges, and the relationship between the experience and the labor market.

5 Evaluation Results

The analysis of the teaching approach is based on the perceptions of the 32 students who completed the course, out of the 35 initially enrolled, all of whom answered the questionnaire anonymously at the end of the semester (100% of completing students). The results focus on the students' experience with the course as a whole and with the specific active learning methods applied, and are organized according to the research questions defined in the GQM framework.

5.1 Context Factors, Motivation, and Participants' Profile

Before presenting the results of the research questions, it is important to contextualize the profile and prior motivation of the participants. The background section on context factors collected information on previous professional and academic experience and on students' familiarity with Software Engineering practices, as well as their initial perception of the relevance of the field.

As for prior experience, 40.6% of respondents (13 students) reported never having had practical experience with any of the listed items, indicating that the cohort was predominantly composed of beginners. The remaining 59.4% reported some level of experience, with the most common being participation in team software development (34.4%) and individual development of systems (31.3%), as shown in Figure 1.

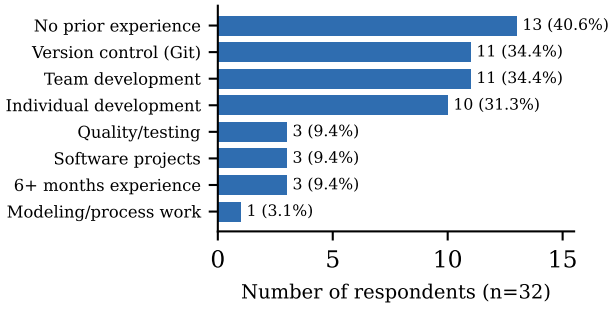

Figure 1: Participants' prior experience.

With respect to familiarity with the specific topics of the course, the data reveal that most students had only initial or superficial contact with the subjects. In Software Process Models, 43.8% reported having heard of the topic without ever using it, and 40.6% had used it superficially. For version control with Git, students showed greater prior maturity in this particular topic. Figure 2 details students' familiarity with each topic of the course.

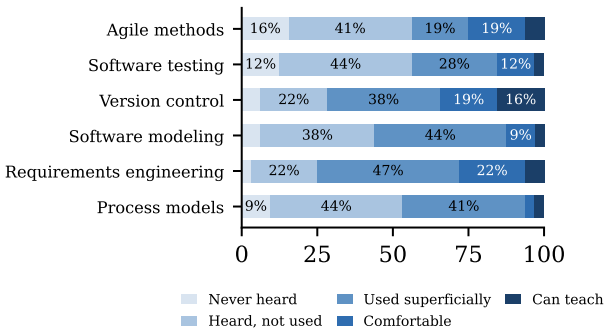

Figure 2: Familiarity with SE practices.

Turning to the perceived importance of Software Engineering, 100% of the students agreed that the field is important for software

development, with 78.1% expressing strong agreement. Additionally, 87.5% reported being motivated to learn more about the course topics, with only 12.5% adopting a neutral position and no negative responses. These data suggest that the class held a favorable predisposition toward learning, which contributes to the internal validity of the study.

5.2 RQ1: Adequacy in Terms of Relevance, Sufficiency, and Teaching Method

RQ1 aimed to evaluate students' perception of the adequacy of the active learning methods applied in the course across three dimensions: content relevance, content sufficiency for understanding the topics, and adequacy of the teaching method employed. Responses were collected on a five-point Likert scale (0, Strongly disagree, to 4, Strongly agree).

Content Relevance. With respect to the relevance of the topics taught, results were broadly positive across all thematic areas of the course. The Requirements Engineering content received 100% positive ratings: 62.5% of students answered "Strongly agree" and 37.5% answered "Agree." Similarly, the topics of Software Process Models and Software Modeling and Design were also rated highly (93.8% positive each), with only two students adopting a neutral position in each category. Software Quality and Testing (78.1% positive) and Configuration Management (68.8% positive) showed more moderate positive rates, with 7 and 10 students in a neutral position, respectively. No negative responses were recorded for the relevance variable in any of the five topics, indicating that students broadly recognized academic and practical value in all the content addressed throughout the semester.

Content Sufficiency. Regarding the sufficiency of content for understanding the topics, the results were also predominantly positive, although with more pronounced variation across topics. Requirements Engineering scored highest, with 90.7% positive ratings (46.9% "Strongly agree" and 43.8% "Agree"). Software Process Models (75.0% positive) and Software Modeling and Design (78.1% positive) also produced satisfactory results. The topic with the lowest perceived sufficiency was Configuration Management, with only 53.1% positive ratings and 12 students (37.5%) in a neutral position, plus 3 disagreements (9.4%). Software Quality and Testing also recorded 3 disagreements, with 71.9% positive ratings and 6 neutrals. These results indicate that, although students recognize the relevance of these topics, some considered the time and materials provided to be insufficient for an in-depth understanding.

Teaching Method. For this dimension, which assessed whether the active learning methods used were perceived as adequate in each thematic unit, the results followed a pattern similar to that observed for content sufficiency. Requirements Engineering again stood out with 90.6% positive ratings, with half of the students (50.0%) choosing "Strongly agree." Software Process Models (81.2% positive) and Modeling and Design (78.1% positive) also produced satisfactory results. These three topics consistently received the highest ratings across the three adequacy dimensions, suggesting that the applied dynamics were especially well received. Configuration Management and Software Quality and Testing presented the least expressive results for teaching method, with 62.5% positive ratings each. In Configuration Management, 37.5% of students adopted a neutral position, although no disagreements were recorded. In Quality and Testing, 34.4% were neutral and 1 disagreement was registered. The concentration of neutral responses in these two topics suggests that the use of active learning methods in these units can be refined in future editions of the course to increase engagement and clarity.

5.3 RQ2 and RQ3: Motivation, Attractiveness, and Theory–Practice Integration

RQ2 and RQ3 aimed to evaluate whether the use of active learning methods throughout the course was perceived by students as motivating and adequate for theory–practice integration. To this end, responses were collected on the attractiveness dimensions: course planning, motivation, theory–practice integration, time adequacy, complexity, creativity, and the playful–challenging character of the dynamics.

Results show that most students rated all attractiveness dimensions positively, as summarized in Figure 3. The course was perceived as attractively planned by 93.8% of respondents (50.0% "Strongly agree" and 43.8% "Agree"), with no negative ratings.

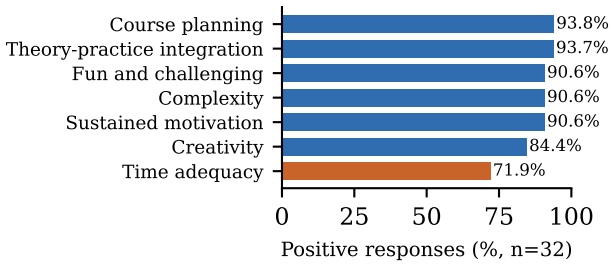

Figure 3: Positive responses (%) across the seven attractiveness dimensions. Time adequacy stands out as the only dimension below 80%.

Sustained motivation (90.6%), theory–practice integration (93.7%, with 53.1% in strong agreement), complexity (90.6%), and the fun and challenging character (90.6%) were all rated highly, as shown in Figure 3. Creativity (84.4%) showed greater polarization, with 2 negative and 3 neutral responses.

The only item with a more modest result was the adequacy of time for carrying out the activities, with 71.9% positive ratings. On this item, 12.5% of students disagreed with the sufficiency of the available time. This finding is consistent with the analysis of open-ended responses (RQ4), in which time shortage was the most recurring weakness.

5.4 RQ4: Strengths and Weaknesses of the Approach

RQ4 sought to collect students' qualitative perceptions of the positive and negative aspects of the teaching experience. The responses to the open-ended questions were analyzed and categorized by thematic similarity. Table 3 presents the identified strengths and weaknesses, with the frequency of mentions for each category.

Table 3: Strengths and weaknesses cited by the participants.

Strengths	Students
Simulation of a real work environment	12
Teamwork and collaborative learning	10
Theory–practice integration	6
Motivation and engagement	5
Use of active learning dynamics	4
Creative freedom and autonomy	3
Good organization and planning	3
Weaknesses	**Students**
Insufficient time or classes for depth	9
Heavy workload and short deadlines	5
Student dropout affected the teams	2
Bureaucratic/theoretical difficulty	2
Need more contact with the clients	1
No code implementation in the projects	1

The most recurring strengths highlight what students most valued in the use of active learning methods: the simulation of an environment close to the labor market, with students playing client and developer roles throughout the semester. Several students stressed that this experience exposed them to real situations such as requirement changes, the need to communicate with the client, and the use of professional tools such as GitHub and technical documentation.

Among the reported weaknesses, the most cited was the limitation of time, both due to canceled classes and to the perception that the semester was not enough to go in depth on all the content. This result is consistent with the quantitative data on the attractiveness dimension concerning the time of the dynamics, on which 12.5% of students disagreed with time adequacy. Additionally, two students mentioned the dropout of classmates during the semester as harmful to team engagement, an observation consistent with the fact that 3 of the 35 enrolled students left the course before the end of the term, an aspect that represents an inherent challenge of methods based on teamwork.

Finally, 87.5% (28 of the 32 respondents) believe that other courses could benefit from the use of similar active learning methods, with justifications mostly centered on the approximation to industry realities, collaborative learning, and the higher engagement that these methods provide.

5.5 Cross-cutting Observations

To support a consolidated reading of the results presented above, Table 4 summarizes the positive-response percentages for each course unit across the three adequacy dimensions (relevance, sufficiency, and teaching method). Two patterns emerge from this view. First, the three units that are most tightly woven into the simulated startup narrative, namely Requirements Engineering, Process Models, and Modeling and Design, cluster at the top of the table, with positive rates at or above 75% on every dimension and reaching 100% on relevance. Second, the two units that operate transversally to the project flow, Configuration Management (carried out continuously across the semester) and Quality and Testing (introduced as a discrete activity), concentrate the moderate and neutral responses, with the lowest scores on sufficiency (53.1% and 71.9%) and on teaching method (62.5% on both).

Table 4: Positive responses (%) per course unit across the three adequacy dimensions.

Course unit	Relevance	Sufficiency	Method
Requirements Engineering	100.0	90.7	90.6
Process Models	93.8	75.0	81.2
Modeling and Design	93.8	78.1	78.1
Quality and Testing	78.1	71.9	62.5
Configuration Management	68.8	53.1	62.5

This reinforces, quantitatively, that the depth of integration between each unit and the simulated workflow appears to mediate perceived sufficiency and method adequacy: the narrative-driven units, with a direct artifact per project step, are perceived more positively, whereas the transversal units, not anchored to a single deliverable, draw the neutral responses that warrant the improvements discussed in Section 6.

6 Discussion

The results obtained show that adopting active learning methods in Software Engineering I, through the simulation of a startup ecosystem with role inversion, yielded effects that extend beyond academic performance and shape how students perceive the value and overall experience of their learning. The qualitative analysis of responses and sentiments reveals increased engagement and a shift in how students think and act when facing SE challenges, echoing what prior PBL studies in SE have already observed [15, 16].

For RQ1, the relevance and sufficiency scores above 90% for Requirements Engineering, Process Models, and Modeling reflect the direct effect of immersion in realistic situations of elicitation, negotiation, and scope-change management, as advocated by Daun et al. [7] as a consolidated trend of the past decade in RE teaching and corroborated by Macedo et al. [10] in a remote role-playing context. These figures are also comparable to those reported by Colares et al. [6] in SPI teaching, where 90% of students in the experimental group strongly agreed with the relevance of the content, suggesting that active learning methods, when anchored in consolidated industrial practices, produce consistent acceptance rates across different SE areas. The convergence between these studies is relevant because it broadens the body of evidence on the effectiveness of active learning methods, extending it beyond SPI alone and isolated RE toward an integrated SE ecosystem.

By contrast, Configuration Management and Software Quality and Testing produced more moderate scores, with a concentration of neutral responses. Three factors help explain this pattern. First, students' lack of prior familiarity with Git/GitHub imposed a learning curve parallel to the conceptual content, splitting attention between mastering the tool and absorbing the topic. Second, the testing and review dynamics were one-off and did not connect to the continuous flow of the project, in contrast to the Requirements and Process units, which had direct correspondence with the

simulation phases. Third, the chronological position of these topics in the semester coincided with the period of greatest pressure on deliverables, reducing the room for reflective experimentation. By naming these likely causes explicitly, an apparently neutral result becomes an actionable agenda for future editions, consistent with Cico et al. [4]'s recommendation on the need to treat instructional-design dimensions separately within startup-based teaching frameworks.

Turning to RQ2 and RQ3, the apparent contradiction between high attractiveness (more than 90% positive ratings for motivation, theory–practice integration, and complexity) and the moderate adequacy of time (71.9% positive, 12.5% disagreements) reveals a structural tension in the simulation model: the richness of the dynamics that sustains engagement is the same that puts pressure on the schedule. This tension is not a defect, but a characteristic of the paradigm, also observed in other immersive PBL experiences in SE [15, 16]. Strategies to manage it, such as shifting more exploratory tasks to asynchronous work and reducing the number of dynamics in favor of deeper exploration of the most immersive ones, constitute design choices that should guide future replications.

For RQ4, the thematic categorization of open-ended responses converges with the quantitative data and reveals what the Likert metric alone cannot capture. The simulation of a real work environment was the most-mentioned strength (12 mentions), followed by collaborative work (10 mentions) and theory–practice integration (6 mentions). This pattern isolates the bidirectional client–developer inversion as the element students perceived as most distinctive of the approach, beyond the use of active learning methods alone. This finding closely parallels Macedo et al. [10], who reported that role-playing between students in client and developer roles improved communication, analytical reasoning, conflict resolution, and empathy, although their study was restricted to RE. The consistency between the two studies suggests that the perceived gain from role inversion is not limited to elicitation but extends to the full cycle when the inversion is sustained throughout the semester. It also resonates with Colares et al. [6]'s findings in SPI teaching, where sentiment analysis using the Syuzhet algorithm identified trust, joy, and anticipation as predominant emotions, with recurring terms such as "practice," "project," and "market", precisely the same central terms identified in the open-ended responses of the present study. The lexical and emotional overlap between studies with distinct pedagogical objects (SPI, RE, and an integrated SE ecosystem) is significant: it suggests that active learning methods, when anchored in real industrial contexts, activate a consistent professionalizing vocabulary among students, contributing to the early construction of a software-engineer identity.

The finding that 87.5% of respondents consider that other courses would benefit from a similar approach is especially telling, since it indicates transferability perceived by the students themselves, that is, their recognition that the model is not restricted to the scope of the course where it was applied but rather embodies a generalizable pedagogical principle. This finding addresses the gap identified by Daun et al. [7], who, in a systematic review of a decade of research on RE teaching, found that only 10.5% of primary studies consider systematic pedagogical grounding and that most proposals concentrate on isolated areas without evidence of transferability. The notion of a sustained ecosystem articulated by Ståhl et al. [16] as a driver of lasting virtuous impacts in PBL provides theoretical support for the transferability reported here and points to paths for institutionalizing the approach beyond a single course. By reporting both broad quantitative results and the explicit perception of broader applicability, this study contributes to the body of evidence supporting the systematic adoption of active learning methods in SE curricula, especially in institutions seeking to align training with the demands of a market that requires professionals capable of negotiating, collaborating, and adapting to changing contexts.

6.1 Pedagogical Implications and Lessons Learned

The interpretation of the quantitative and qualitative findings discussed so far gains full meaning when confronted with the continuous observation of the researchers throughout the semester. This subsection systematizes, from that angle, the pedagogical implications that emerged from practice, that is, what actually worked, what required real-time adjustment, and what should be reconsidered in future replications, complementing the response to RQ4 with elements not captured by the questionnaire.

What is new in this reading is not each observation in isolation, most of which appears, separately, in prior reports, but what they jointly reveal about sustaining the bidirectional inversion across a full semester, a setting largely absent from studies confined to a single topic or to a few weeks. Under this lens, the central gain is not engagement in general but the client-developer empathy built through the prolonged inversion, the element students valued most, with 12 mentions in the open-ended responses. The same lens explains the difficulties: the units that broke the inversion chain, Configuration Management and Testing, because they were not anchored to a project artifact, are precisely the ones that concentrated neutral responses, and the affinity-based team formation weakened the inversion by producing engagement imbalance and dropout. The lesson specific to this design, therefore, is that the value of role inversion depends on keeping every unit anchored to the shared artifact and on composing teams so that the inversion is sustained throughout the semester, a point resumed in the future-work agenda.

What worked well. The Paper Airplane Factory dynamic, used at the beginning of the course to make the concepts of Waterfall and Agile processes tangible, was received with high engagement and enthusiasm. Being concrete, visual, and competitive, it accelerated the understanding of abstract concepts such as throughput, process bottlenecks, and iterative improvement cycles, topics that, in traditional lectures, are often perceived as detached from reality. This result is consistent with López-Fernández et al. [9]'s findings, who validated, in a study with 242 SE students, that playful and tangible activities for teaching lifecycle models produce high motivation (4.51/5.0) and perceived usefulness (4.55/5.0) among students, while also creating a positive classroom climate (4.75/5.0). The University Restaurant dynamic proved especially effective for teaching Requirements Engineering: by simulating a real client with changing needs, students experienced firsthand the pressure of scope change, the challenge of communicating with the client, and the need to document requirements with precision. The role

inversion, in which each group simultaneously acted as developer of another team's project and client of its own, was the most praised pedagogical element and the one that generated the greatest empathy among participants.

What required real-time adjustment. Time management within the dynamics required continuous calibration throughout the semester. In particular, the UML modeling and GitHub repository configuration phases took more time than planned, since a significant share of students lacked prior familiarity with these tools. The solution adopted was to relax intermediate deadlines and provide asynchronous support tutorials, which eased pressure without compromising the overall project flow. In addition, initially forming teams by affinity rather than by technical criteria or profile diversity produced imbalances in engagement across teams over the semester, a factor that contributed to the student dropouts reported among the RQ4 weaknesses. This critical point echoes the findings of Tamayo Avila et al. [17], whose ASEST+ framework, validated in a quasi-experiment with Scrum teams, shows that team cohesion in agile SE education is strongly influenced by antecedents such as personality traits, conflict resolution, and task interdependence, suggesting that team composition should be treated as a pedagogical design variable rather than as an incidental decision delegated to students. Accordingly, in future editions the affinity-based grouping is to be replaced by objective composition criteria, without fully removing student autonomy: the diversity of prior performance, drawn from the familiarity self-assessment already collected in the context questionnaire; the distribution of practical experience across teams; and the balance of profiles within each team, so that no team concentrates the more experienced or the more novice students. These criteria act on the antecedents of cohesion identified by ASEST+, so that cohesion is fostered by design rather than left to chance.

What should be revised in future editions. The Configuration Management and Software Quality and Testing units would benefit from dynamics more tightly integrated with the simulation ecosystem. Whereas the other units had direct correspondence with the startup project phases, these two topics were addressed on a one-off basis, without an equivalent immersive activity. A natural option would be to introduce, in the testing unit, a "peer review" dynamic in which one team tests the product developed by another, simulating the role of a QA team, a strategy consistent with Tiwari et al. [18]'s recommendations on the effectiveness of cross-evaluation as an active learning technique in SE. For Configuration Management, mandating branching strategies and pull requests on GitHub Projects from the beginning of the semester, rather than only during the execution phase, may increase the perceived sufficiency of this content. Finally, we recommend a formal mechanism for team redistribution from the middle of the semester onward, to absorb dropout impacts without penalizing the affected groups, an adjustment specific to our setting rather than part of ASEST+, which fosters cohesion within stable teams [17].

A specific reading is warranted for the neutral responses concentrated in the transversal units. The evidence suggests that the difficulty was less a matter of tooling than of instructional design: operational mastery of the tool and conceptual understanding of the unit competed for the same instructional moment, so that the effort of operating the tool displaced the effort that should have served the concept. Three design decisions are therefore proposed for future editions. First, operational fluency is to be decoupled from conceptual learning and treated as a prerequisite, built through short, low-stakes familiarization activities that precede the conceptual unit and are not assessed together with it. Second, each transversal unit is to be anchored in an artifact the students produced earlier, so that the topic is experienced as a continuation of their own project rather than as a new subject introduced late, a condition that the results associate with neutral responses. Third, the content is to be sequenced as progressively scaffolded tasks, introducing the cognitive demand in increments and gradually withdrawing support, which reserves class time for conceptual discussion rather than for operational troubleshooting.

6.2 Threats to Validity

Validity threats were analyzed along the four dimensions classically proposed for empirical SE research, namely internal, external, construct, and conclusion validity, with the main risks and mitigations for each.

Internal validity concerns whether the reported perceptions stem from the pedagogical approach rather than from uncontrolled factors. Three risks were identified: social-desirability bias, mitigated by anonymous administration with no identifiers; coding bias in the open-ended responses, mitigated by independent coding with supervisor review and joint discussion of divergences; and the absence of a control group, a known limitation of the experience-report design that precludes strict causal attribution, mitigated in part by sizing the findings against previously published studies.

External validity concerns generalization beyond the investigated context. The results come from a single class at the Universidade Federal do Amapá (UNIFAP), in one semester and with a specific profile (intermediate-stage undergraduates), which precludes categorical claims about replicability in different settings, such as larger classes or other programs and infrastructures. This was mitigated by the detailed description of context, profile, dynamics, and instrument, to support transferability by the reader; the 87.5% of perceived transferability reported by students adds evidence that the model is not seen as course-specific.

Construct validity assesses whether the instruments capture the intended constructs (engagement, comprehension, perceived applicability, method adequacy). Two limitations are acknowledged: the five-point Likert scale can flatten nuances that only open-ended responses capture, and constructs such as "perceived transferability" and "motivation" are self-reports open to distinct interpretations. These were mitigated by aligning each questionnaire item with a GQM metric tied to an a priori objective and by triangulating Likert and content-analysis data, so that interpretations rest on convergent sources rather than a single metric.

Conclusion validity concerns the credibility of the inferences. The moderate sample (n=32) limits statistical power, which is why the work confines itself to a descriptive and interpretive analysis, as befits an experience report. A possible novelty effect, since the approach departs markedly from the traditional teaching students were used to, may have inflated part of the enthusiasm; as mitigation, the consistency across independent sources (Likert

data, categorized open responses, and convergence with Colares et al. [6]) strengthens the qualitative conclusions, and the explicit critical points (Configuration Management, Testing, time adequacy) show the findings were not uniformly positive.

7 Conclusion

This experience report investigated how undergraduate students perceive the use of active learning methods in a mandatory Software Engineering course, applied through a simulated startup ecosystem in which students alternated between client and developer roles throughout the semester. This bidirectional role inversion is not a contribution in itself, but the mechanism that sustained the in-class dynamics and enabled the collection of the perceptions reported here.

The study makes three contributions: empirical evidence, structured by GQM and triangulating Likert data with content analysis, on how students perceive active learning methods along the full SE lifecycle; a replicable instructional arrangement that integrates the five evaluated units under a single narrative; and the identification of specific weaknesses in Configuration Management and Testing, translating neutral ratings into a concrete agenda for future editions.

The findings should be read in light of the main limitations, a small sample (n=32), a single-class design, and the absence of longitudinal follow-up. Future work will replicate the evaluation in other courses to test the reported transferability, investigate the neutral ratings of Configuration Management and Testing through interviews, and refine the GQM instrument for reuse in other courses and institutions.

Artifact Availability

The questionnaire and the anonymized response data are available at https://zenodo.org/records/21415875 under the Creative Commons CC BY 4.0 license.

References

[1] ACM/IEEE-CS Joint Task Force on Computing Curricula. 2014. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. https://www.acm.org/binaries/content/assets/education/se2014.pdf.

[2] L. Bardin. 2011. *Análise de Conteúdo* (3 ed.). Edições 70, Lisboa.

[3] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. John Wiley & Sons.

[4] Orges Cico, Anh Nguyen Duc, and Letizia Jaccheri. 2021. The development and validation of a framework for teaching software engineering through startup practice in higher education. In *2021 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9. doi:10.1109/FIE49875.2021.9637256

[5] A. F. de O. Colares, J. C. C. Furtado, and S. R. B. Oliveira. 2022. Use of Gamification as Implementation Approach for Software Process Improvement: Trends and Gaps. In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*.

[6] A. F. de O. Colares, S. R. B. Oliveira, and J. C. C. Furtado. 2025. A Qualitative Evaluation of an Experiment on the Application of Active Methodologies in Teaching Software Process Improvement. In *Anais do XXIV Simpósio Brasileiro de Qualidade de Software (SBQS)*. SBC, São José dos Campos, SP, Brasil.

[7] M. Daun, A. M. Grubb, and V. Stenkova. 2022. A systematic literature review of requirements engineering education. *Requirements Engineering* 28, 2 (2022), 145–175. doi:10.1007/s00766-022-00381-9

[8] Anderson dos Santos Guerra and Julio Cezar Costa Furtado. 2019. A Practical Approach to Teaching Requirements Engineering in Computing Programs. In *ICSEA 2019: The Fourteenth International Conference on Software Engineering Advances*. IARIA, Valencia, Spain, 36–43.

[9] Daniel López-Fernández, Aldo Gordillo, and Fernando Ortega. 2021. LEGO Serious Play in Software Engineering Education. *IEEE Access* 9 (2021), 103120–103131. doi:10.1109/ACCESS.2021.3095552

[10] Gretchen Macedo, Awdren Fontão, and Bruno Gadelha. 2024. Building soft skills through a role-play based approach for Requirements Engineering remote education. *Journal of the Brazilian Computer Society* 30, 1 (2024), 1–16. doi:10.5753/jbcs.2024.3071

[11] Daniel Moritz Marutschke, Victor V. Kryssanov, and Patricia Brockmann. 2020. Teaching Distributed Requirements Engineering: Simulation of an Offshoring Project with Geographically Separated Teams. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*. IEEE, 1–5. doi:10.1109/CSEET49119.2020.9206178

[12] Michael Prince. 2004. Does Active Learning Work? A Review of the Research. *Journal of Engineering Education* 93, 3 (2004), 223–231. doi:10.1002/j.2168-9830.2004.tb00809.x

[13] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (2009), 131–164. doi:10.1007/s10664-008-9102-8

[14] M. Serrano, M. Serrano, and F. Napolitano. 2008. Avaliação Experimental de um Método para Avaliação de Equipes de Requisitos. In *Anais do XXII Simpósio Brasileiro de Engenharia de Software (SBES)*. SBC, Campinas, SP, Brasil. doi:10.5753/sbes.2008.21328

[15] Maurício Ronny de Almeida Souza, Renata Moreira, and Eduardo Figueiredo. 2019. Students Perception on the use of Project-Based Learning in Software Engineering Education. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES)*. ACM, 537–546. doi:10.1145/3350768.3352457

[16] Daniel Ståhl, Kristian Sandahl, and Lena Buffoni. 2022. An Eco-System Approach to Project-Based Learning in Software Engineering Education. *IEEE Transactions on Education* 65, 4 (2022), 514–523. doi:10.1109/TE.2021.3137344

[17] Daymy Tamayo Avila, Wim Van Petegem, and Monique Snoeck. 2022. Improving Teamwork in Agile Software Engineering Education: The ASEST+ Framework. *IEEE Transactions on Education* 65, 1 (2022), 18–29. doi:10.1109/TE.2021.3084095

[18] Saurabh Tiwari, Deepti Ameta, and Paramvir Singh. 2018. Teaching requirements engineering concepts using case-based learning. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training*. ACM, 8–15. doi:10.1145/3194779.3194791

[19] S. Tockey. 2015. Insanity, Hiring, and the Software Industry. *Computer* 48, 11 (2015), 96–101. doi:10.1109/MC.2015.318